Attorney Docket No. BRCMP005

# PATENT APPLICATION

## CLASSIFICATION ENGINE IN A CRYPTOGRAPHY ACCELERATION CHIP


Inventors: Suresh Krishna

Christopher Owen

Derrick Lin

Joe Tardo

Patrick Law

Phillip Smith

Assignee: Broadcom Corporation
Irvine, CA

BEYER WEAVER & THOMAS, LLP
P.O. Box 130
Mountain View, CA 94042-0130
Telephone (510) 843-6200

# CLASSIFICATION ENGINE IN A CRYPTOGRAPHY ACCELERATION CHIP

*By Inventors: Suresh Krishna, Christopher Owen, Derrick Lin, Joe Tardo, Patrick Law and Phillip Smith*

## CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims priority from U.S. Provisional Application No. 60/142,870, entitled NETWORKING SECURITY CHIP ARCHITECTURE AND IMPLEMENTATIONS FOR CRYPTOGRAPHY ACCELERATION, filed July 8, 1999; and claims priority from U.S. Provisional Application No. 60/159,~~012~~ 511, entitled UBIQUITOUS BROADBAND SECURITY CHIP, filed October 12, 1999, the disclosures of which are herein incorporated by reference herein for all purposes.

This application is related to concurrently filed U.S. Application No. _____ (Atty. Docket No. BRCMP003), entitled DISTRIBUTED PROCESSING IN A CRYPTOGRAPHY ACCELERATION CHIP, the disclosure of which is incorporated by reference herein for all purposes.

## BACKGROUND OF THE INVENTION

The present invention relates generally to the field of cryptography, and more particularly to an architecture and method for cryptography acceleration.

Many methods to perform cryptography are well known in the art and are discussed, for example, in Applied Cryptography, Bruce Schneier, John Wiley & Sons, Inc. (1996, 2nd Edition), herein incorporated by reference. In order to improve the speed of cryptography processing, specialized cryptography accelerator chips have been developed. For example, the Hi/fn ™ 7751 and the VLSI ™ VMS115 chips provide hardware cryptography acceleration that out-performs similar software implementations. Cryptography accelerator chips may be included in routers or gateways, for example, in order to provide automatic IP packet encryption/decryption. By embedding cryptography functionality in network hardware, both system performance and data security are enhanced.

However, these chips require sizeable external attached memory in order to operate. The VLSI VMS115 chip, in fact, requires attached synchronous SRAM, which is the most expensive type of memory. The substantial additional memory requirements make these solutions unacceptable in terms of cost versus performance for many applications.

Also, the actual sustained performance of these chips is much less than peak throughput that the internal cryptography engines (or "crypto engines") can sustain. One

reason for this is that the chips have a long "context" change time. In other words, if the cryptography keys and associated data need to be changed on a packet-by-packet basis, the prior art chips must swap out the current context and load a new context, which reduces the throughput. The new context must generally be externally loaded from software, and for many applications, such as routers and gateways that aggregate bandwidth from multiple connections, changing contexts is a very frequent task.

Moreover, the architecture of prior art chips does not allow for the processing of cryptographic data at rates sustainable by the network infrastructure in connection with which these chips are generally implemented. This can result in noticeable delays when cryptographic functions are invoked, for example, in e-commerce transactions.

Recently, an industry security standard has been proposed that combines both "DES/3DES" encryption with "MD5/SHA1" authentication, and is known as "IPSec." By incorporating both encryption and authentication functionality in a single accelerator chip, over-all system performance can be enhanced. But due to the limitations noted above, the prior art solutions do not provide adequate performance at a reasonable cost.

Thus it would be desirable to have a cryptography accelerator chip architecture that is capable of implementing the IPSec specification (or any other cryptography standard), at much faster rates than are achievable with current chip designs.

## SUMMARY OF THE INVENTION

In general, the present invention provides an architecture for a cryptography accelerator chip that allows significant performance improvements over previous prior art designs. In various embodiments, the architecture enables parallel processing of packets through a plurality of cryptography engines and includes a classification engine configured to efficiently process encryption/decryption of data packets. Cryptography acceleration chips in accordance may be incorporated on network line cards or service modules and used in applications as diverse as connecting a single computer to a WAN, to large corporate networks, to networks servicing wide geographic areas (e.g., cities). The present invention provides improved performance over the prior art designs, with much reduced local memory requirements, in some cases requiring no additional external memory. In some embodiments, the present invention enables sustained full duplex Gigabit rate security processing of IPSec protocol data packets.

In one aspect, the present invention provides a cryptography acceleration chip having a classification engine configured to receive a complete IP packet and determine what keys are needed to encrypt or decrypt the packet.

In another aspect, the invention provides a method for accelerating cryptography processing of data packets. The method involves receiving a plurality of complete data packets on a cryptography acceleration chip and processing the data packets with a classification engine to determine what keys are needed to encrypt or decrypt the packets.

These and other features and advantages of the present invention will be presented in more detail in the following specification of the invention and the accompanying figures which illustrate by way of example the principles of the invention.

## BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be readily understood by the following detailed description in conjunction with the accompanying drawings, wherein like reference numerals designate like structural elements, and in which:

Figs. 1A and B are high-level block diagrams of systems implementing a cryptography accelerator chip in accordance with one embodiment the present invention.

Fig. 2 is a high-level block diagram of a cryptography accelerator chip in accordance with one embodiment the present invention.

Fig. 3 is a block diagram of a cryptography accelerator chip architecture in accordance with one embodiment of the present invention.

Fig. 4 is a block diagram illustrating a DRAM-based or SRAM-based packet classifier in accordance with one embodiment the present invention.

Fig. 5 is a block diagram illustrating a CAM-based packet classifier in accordance with one embodiment the present invention.

Figs. 6A and 6B are flowcharts illustrating aspects of inbound and outbound packet processing in accordance with one embodiment the present invention.

Fig. 7 shows a block diagram of a classification engine in accordance with one embodiment of the present invention, illustrating its structure and key elements.

## DETAILED DESCRIPTION OF THE INVENTION

Reference will now be made in detail to some specific embodiments of the invention including the best modes contemplated by the inventors for carrying out the invention. Examples of these specific embodiments are illustrated in the accompanying drawings. While the invention is described in conjunction with these specific embodiments, it will be understood that it is not intended to limit the invention to the described embodiments. On the contrary, it is intended to cover alternatives, modifications, and equivalents as may be included within the spirit and scope of the invention as defined by the appended claims. In the following description, numerous specific details are set forth in order to provide a thorough understanding of the present invention. The present invention may be practiced without some or all of these specific details. In other instances, well known process operations have not been described in detail in order not to unnecessarily obscure the present invention.

In general, the present invention provides an architecture for a cryptography accelerator chip that allows significant performance improvements over previous prior art designs. In preferred embodiments, the chip architecture enables "cell-based" processing of random-length IP packets, as described in copending U.S. Patent Application No. 09/510,486, entitled SECURITY CHIP ARCHITECTURE AND IMPLEMENTATIONS FOR CRYPTOGRAPHY ACCELERATION, incorporated by reference herein in its entirety for all purposes. Briefly, cell-based packet processing involves the splitting of IP packets, which may be of variable and unknown size, into smaller fixed-size "cells." The fixed-sized cells are then processed and reassembled (recombined) into packets. The cell-based packet processing architecture of the present invention allows the implementation of a processing pipeline that has known processing throughput and timing characteristics, thus making it possible to fetch and process the cells in a predictable time frame. In preferred embodiments, the cells may be fetched ahead of time (pre-fetched) and the pipeline may be staged in such a manner that the need for attached (local) memory to store packet data or control parameters is minimized or eliminated.

Moreover, in various embodiments, the architecture enables parallel processing of packets through a plurality of cryptography engines, for example four, and includes a classification engine configured to efficiently process encryption/decryption of data packets. Cryptography acceleration chips in accordance may be incorporated on network line cards or service modules and used in applications as diverse as connecting a single computer to a WAN, to large corporate networks, to networks servicing wide geographic areas (e.g., cities). The present invention provides improved performance over the prior art designs, with much reduced local memory requirements, in some cases requiring no additional external memory.

In some embodiments, the present invention enables sustained full duplex Gigabit rate security processing of IPSec protocol data packets.

In this specification and the appended claims, the singular forms "a," "an," and "the" include plural reference unless the context clearly dictates otherwise. Unless defined otherwise, all technical and scientific terms used herein have the same meaning as commonly understood to one of ordinary skill in the art to which this invention belongs.

The present invention may be implemented in a variety of ways. Figs. 1A an 1B illustrate two examples of implementations of the invention as a cryptography acceleration chip incorporated into a network line card or a system module, respectively, in a standard processing system in accordance with embodiments of the present invention.

As shown in Fig. 1A, the cryptography acceleration chip 102 may be part of an otherwise standard network line card 103 which includes a WAN interface 112 that connects the processing system 100 to a WAN, such as the Internet, and manages in-bound and out-bound packets. The chip 102 on the card 103 may be connected to a system bus 104 via a standard system interface 106. The system bus 104 may be, for example, as standard PCI bus, or it may be a high speed system switching matrix, as are well known to those of skill in the art. The processing system 100 includes a processing unit 114, which may be one or more processing units, and a system memory unit 116.

The cryptography acceleration chip 102 on the card 103 also has associated with it a local processing unit 108 and local memory 110. As will be described in more detail below, the local memory 110 may be RAM or CAM and may be either on or off the chip 102. The system also generally includes a LAN interface (not shown) which attaches the processing system 100 to a local area network and receives packets for processing and writes out processed packets to the network.

According to this configuration, packets are received from the LAN or WAN and go directly through the cryptography acceleration chip and are processed as they are received from or are about to be sent out on the WAN, providing automatic security processing for IP packets.

In some preferred embodiments the chip features a streamlined IP packet-in/packet-out interface that matches line card requirements in ideal fashion. As described further below, chips in accordance with the present invention may provide distributed processing intelligence that scales as more line cards are added, automatically matching up security processing power with overall system bandwidth. In addition, integrating the chip onto line cards preserves precious switching fabric bandwidth by pushing security processing to the

edge of the system. In this way, since the chip is highly autonomous, shared system CPU resources are conserved for switching, routing and other core functions.

One beneficial system-level solution for high-end Web switches and routers is to integrate a chip in accordance with the present invention functionality with a gigabit Ethernet MAC and PHY. The next generation of firewalls being designed today require sustained security bandwidths in the gigabit range. Chips in accordance with the present invention can deliver sustained full duplex multi-gigabit IPSec processing performance.

As shown in Fig. 1B, the cryptography acceleration chip 152 may be part of a service module 153 for cryptography acceleration. The chip 152 in the service module 153 may be connected to a system bus 154 via a standard system interface 156. The system bus 154 may be, for example, a high speed system switching matrix, as are well known to those of skill in the art. The processing system 150 includes a processing unit 164, which may be one or more processing units, and a system memory unit 166.

The cryptography acceleration chip 152 in the service module 153 also has associated with it a local processing unit 158 and local memory 160. As will be described in more detail below, the local memory 160 may be RAM or CAM and may be either on or off the chip 152. The system also generally includes a LAN interface which attaches the processing system 150 to a local area network and receives packets for processing and writes out processed packets to the network, and a WAN interface that connects the processing system 150 to a WAN, such as the Internet, and manages in-bound and out-bound packets. The LAN and WAN interfaces are generally provided via one or more line cards 168, 170. The number of line cards will vary depending on the size of the system. For very large systems, there may be thirty to forty or more line cards.

According to this configuration, packets received from the LAN or WAN are directed by the high speed switching matrix 154 to memory 166, from which they are sent to the chip 152 on the service module 153 for security processing. The processed packets are then sent back over the matrix 154, through the memory 166, and out to the LAN or WAN, as appropriate.

Basic Features, Architecture and Distributed Processing

Fig. 2 is a high-level block diagram of a cryptography chip architecture in accordance with one embodiment of the present invention. The chip 200 may be connected to external systems by a standard PCI interface (not shown), for example a 32-bit bus operating at up to 33 MHz. Of course, other interfaces and configurations may be used, as is well known in the art, without departing from the scope of the present invention.

Referring to Fig. 2, the IP packets are read into a FIFO (First In First Out buffer) input unit 202. This interface (and the chip's output FIFO) allow packet data to stream into and out of the chip. In one embodiment, they provide high performance FIFO style ports that are unidirectional, one for input and one for output. In addition, the FIFO 202 supports a bypass capability that feeds classification information along with packet data. Suitable FIFO-style interfaces include GMII as well as POS-PHY-3 style FIFO based interfaces, well known to those skilled in the art.

From the input FIFO 202, packet header information is sent to a packet classifier unit 204 where a classification engine rapidly determines security association information required for processing the packet, such as encryption keys, data, etc. As described in further detail below with reference to Figs. 4, 5 and 6A and B, the classification engine performs lookups from databases stored in associated memory. The memory may be random access memory (RAM), for example, DRAM or SSRAM, in which case the chip includes a memory controller 212 to control the associated RAM. The associated memory may also be contact addressable memory (CAM), in which case the memory is connected directly with the cryptography engines 216 and packet classifier 204, and a memory controller is unnecessary. The associated memory may be on or off chip memory. The security association information determined by the packet classifier unit 204 is sent to a packet distributor unit 206.

The distributor unit 206 determines if a packet is ready for IPSec processing, and if so, distributes the security association information (SA) received from the packet classifier unit 204 and the packet data among a plurality of cryptography processing engines 124, in this case four, on the chip 200, for security processing. This operation is described in more detail below.

The cryptography engines may include, for example, "3DES-CBC/DES X" encryption/decryption "MD5/SHA1" authentication/digital signature processing and compression/decompression processing. It should be noted, however, that the present architecture is independent of the types of cryptography processing performed, and additional cryptography engines may be incorporated to support other current or future cryptography algorithms. Thus, a further discussion of the cryptography engines is beyond to scope of this disclosure.

Once the distributor unit 206 has determined that a packet is ready for IPSec processing, it will update shared IPSec per-flow data for that packet, then pass the packet along to one of the four cryptography and authentication engines 214. The distributor 206 selects the next free engine in round-robin fashion within a given flow. Engine output is also read in the same round-robin order. Since packets are retired in a round-robin fashion that matches their order of issue packet ordering is always maintained within a flow ("per flow

ordering"). For the per-flow ordering case, state is maintained to mark the oldest engine (first one issued) for each flow on the output side, and the newest (most recently issued) engine on the input side; this state is used to select an engine for packet issue and packet retiring. The chip has an engine scheduling module which allows new packets to be issued even as previous packets from the same flow are still being processed by one or more engines. In this scenario, the SA Buffers will indicate a hit (SA auxiliary structure already on-chip), shared state will be updated in the on-chip copy of the SA auxiliary structure, and the next free engine found in round-robin order will start packet processing.

Thus, the distributor 206 performs sequential portions of IPSec processing that rely upon packet-to-packet ordering, and hands off a parallelizable portion of IPSec to the protocol and cryptography processing engines. By providing multiple cryptography engines and processing data packets in parallel chips in accordance with the present invention are able to provide greatly improved security processing performance. The distributor also handles state cleanup functions needed to properly retire a packet (including ensure that packet ordering is maintained) once IPSec processing has completed.

Per-flow ordering offers a good trade-off between maximizing end-to-end system performance (specifically desktop PC TCP/IP stacks), high overall efficiency, and design simplicity. In particular, scenarios that involve a mix of different types of traffic such as voice-over-IP (VoIP), bulk ftp/e-mail, and interactive telnet or web browsing will run close to 100% efficiency. Splitting, if necessary, a single IPSec tunnel into multiple tunnels that carry unrelated data can further enhance processing efficiency.

Per-flow IPSec data includes IPSec sequence numbers, anti-replay detection masks, statistics, as well as key lifetime statistics (time-based and byte-based counters). Note that some of this state cannot be updated until downstream cryptography and authentication engines have processed an entire packet. An example of this is the anti-replay mask, which can only be updated once a packet has been established as a valid, authenticated packet. In one embodiment, the distributor 206 handles these situations by holding up to eight copies of per-flow IPSec information on-chip, one copy per packet that is in process in downstream authentication and crypto engines (each engine holds up to two packets due to internal pipelining). These copies are updated once corresponding packets complete processing.

This scheme will always maintain ordering among IPSec packets that belong to a given flow, and will correctly process packets under all possible completion ordering scenarios.

In addition, in some embodiments, a global flag allows absolute round robin sequencing, which maintains packet ordering even among different flows ("strong ordering").

Strong ordering may be maintained in a number of ways, for example, by assigning a new packet to the next free cryptography processing unit in strict round-robin sequence. Packets are retired in the same sequence as units complete processing, thus ensuring order maintenance. If the next engine in round-robin sequence is busy, the process of issuing new packets to engines is stalled until the engines become free. Similarly, if the next engine on output is not ready, the packet output process stalls. These restrictions ensure that an engine is never "skipped", thus guaranteeing ordering at the expense of some reduced processing efficiency.

Alternatively, strong ordering may be maintained by combining the distributor unit with an order maintenance packet retirement unit. For every new packet, the distributor completes the sequential portions of IPSec processing, and assigns the packet to the next free engine. Once the engine completes processing the packet, the processed packet is placed in a retirement buffer. The retirement unit then extracts processed packets out of the retirement buffer in the same order that the chip originally received the packets, and outputs the processed packets. Note that packets may process through the multiple cryptography engines in out of order fashion; however, packets are always output from the chip in the same order that the chip received them. This is an "out-of-order execution, in-order retirement" scheme. The scheme maintains peak processing efficiency under a wide variety of workloads, including a mix of similar size or vastly different size packets.

Most functions of the distributor are performed via dedicated hardware assist logic as opposed to microcode, since the distributor 206 is directly in the critical path of per-packet processing. The distributor's protocol processor is programmed via on-chip microcode stored in a microcode storage unit 208. The protocol processor is microcode-based with specific instructions to accelerate IPSec header processing.

The chip also includes various buffers 210 for storing packet data, security association information, status information, etc., as described further with reference to Fig. 3, below. For example, fixed-sized packet cells may be stored in payload or packet buffers, and context or security association buffers may be used to store security association information for associated packets/cells.

The output cells are then stored in an output FIFO 216, in order to write the packets back out to the system. The processed cells are reassembled into packets and sent off the chip by the output FIFO 216.

Fig. 3 is a block diagram of a cryptography accelerator chip architecture in accordance with one embodiment of the present invention. The chip 300 includes an input FIFO 302 into which IP packets are read. From the input FIFO 302, packet header information is sent to a

packet classifier unit 204 where a classification engine rapidly determines security association information required for processing the packet, such as encryption keys, data, etc. As described in further detail below, the classification engine performs lookups from databases stored in associated memory. The memory may be random access memory (RAM), for example, DRAM or SSRAM, in which case the chip includes a memory controller 308 to control the associated RAM. The associated memory may also be contact addressable memory (CAM), in which case the memory is connected directly with the cryptography engines 316 and packet classifier 304, and a memory controller is unnecessary. The associated memory may be on or off chip memory. The security association information determined by the packet classifier unit 304 is sent to a packet distributor unit 306 via the chip's internal bus 305.

The packet distributor unit 306 then distributes the security association information (SA) received from the packet classifier unit 304 and the packet data via the internal bus 305 among a plurality of cryptography processing engines 316, in this case four, on the chip 200, for security processing. For example, the crypto engines may include "3DES-CBC/DES X" encryption/decryption "MD5/SHA1" authentication/digital signature processing and compression/decompression processing. As noted above, the present architecture is independent of the types of cryptography processing performed, and a further discussion of the cryptography engines is beyond to scope of this disclosure.

The packet distributor unit 306 includes a processor which controls the sequencing and processing of the packets according to microcode stored on the chip. The chip also includes various buffers associated with each cryptography engine 316. A packet buffer 312 is used for storing packet data between distribution and crypto processing. Also, in this embodiment, each crypto engine 316 has a pair of security association information (SA) buffers 314a, 314b associated with it. Two buffers per crypto engine are used so that one 314b, may hold the SA for a current packet (packet currently being processed) while the other 314a is being preloaded with the security association information for the next packet. A status buffer 310 may be used to store processing status information, such as errors, etc.

Processed packet cells are reassembled into packets and sent off the chip by an output FIFO 318. The packet distributor 306 controls the output FIFO 318 to ensure that packet ordering is maintained.

Packet Classifier

The IPSec cryptography protocol specifies two levels of lookup: Policy (Security Policy Database (SPD) lookup) and Security Association (Security Association Database (SAD) lookup). The policy look-up is concerned with determining what needs to be done

with various types of traffic, for example, determining what security algorithms need to be applied to a packet, without determining the details, e.g., the keys, etc. The Security Association lookup provides the details, e.g., the keys, etc., needed to process the packet according to the policy identified by the policy lookup. The present invention provides chip architectures and methods capable of accomplishing this IPSec function at sustained multiple full duplex gigabit rates.

As noted above, there are two major options for implementing a packet classification unit in accordance with the present invention: CAM based and RAM (DRAM/SSRAM) based. The classification engine provides support for general IPSec policy rule sets, including wild cards, overlapping rules, conflicting rules and conducts deterministic searches in a fixed number of clock cycles. In preferred embodiments, it may be implemented either as a fast DRAM/SSRAM lookup classification engine, or on-chip CAM memory for common situations, with extensibility via off-chip CAM, DRAM or SSRAM. Engines in accordance with some embodiments of the present invention engine are capable of operating at wirespeed rates under any network load. In one embodiment, the classifier processes packets down to 64 bytes at OC12 full duplex rates (1.2Gb/s throughput); this works out to a raw throughput of 2.5M packets per second.

The classifier includes four different modes that allow all IPSec selector matching operations to be supported, as well as general purpose packet matching for packet filtering purposes, for fragment re-assembly purposes, and for site blocking purposes. The classifier is not intended to serve as a general-purpose backbone router prefix-matching engine. As noted above, the classifier supports general IPSec policies, including rules with wildcards, ranges, and overlapping selectors. Matching does not require a linear search of overlapping rules, but instead occurs in a deterministic number of clock cycles.

Security and filtering policies are typically specified using flexible rule sets that allow generic matching to be performed on a set of broad packet selector fields. Individual rules support wildcard specification and ranges for matching parameters. In addition, multiple rules are allowed to overlap, and order-based matching is used to select the first applicable rule in situations where multiple rules apply.

Rule overlap and ordered matching add a level of complexity to hardware-based high-speed rule matching implementations. In particular, the requirement to select among multiple rules that match based on the order in which these rules are listed precludes direct implementation via high-speed lookup techniques that immediately find a matching rule independent of other possible matches.

Chips in accordance with the present invention provide a solution to the problem of matching in a multiple overlapping order-sensitive rule set environment involving a combination of rule pre-processing followed by direct high-speed hardware matching, and supports the full generality of security policy specification languages.

A pre-processing de-correlation step handles overlapping and possibly conflicting rule sets. This de-correlation algorithm produces a slightly larger equivalent rule set that involves zero intersection The new rule set is then implemented via high-speed hardware lookups. High performance algorithms that support incremental de-correlation are available in the art. Where CAM is used, a binarization step is used to convert range-based policies into mask-based lookups suitable for CAM arrays.

The function of the packet classifier is to perform IPSec-specified lookup as well as IP packet fragmentation lookup. These lookups are used by the distributor engine, as well as by the packet input engine (FIFO). In one embodiment, classification occurs based on a flexible set of selectors as follows:

- Quintuple of <src IP addr, dst IP addr, src port, dst port, protocol> → 104 bits match field
- Triple of <src IP addr, dst IP addr, IPSec SPI security parameter index> → 96-bit match field
- Basic match based on <src IP addr, dst IP addr, protocol> → 72-bit match field
- Fragment match based on <src IP, dst IP, fragment ID, protocol> → 88-bit match field

The result of packet classification is a classification tag. This structure holds IPSec security association data and per-flow statistics.

As noted above, a classifier in accordance with the present invention can be implemented using several different memory arrays for rule storage; each method involves various cost/performance trade-offs. The main implementations are external CAM-based policy storage; on-chip CAM-based policy storage; and external RAM (DRAM, SGRAM, SSRAM) based storage. Note that RAM-based lookups can only match complete (i.e. exact) sets of selectors, and hence tend to require more memory and run slower than CAM-based approaches. On-chip CAM offers an attractive blend of good capacity, high performance and low cost.

A preferred approach for cost-insensitive versions of a cryptography acceleration chip in accordance with the present invention is to implement an on-chip CAM and to provide a method to add more CAM storage externally. Rule sets tend to be relatively small (dozens of entries for a medium corporate site, a hundred entries for a large site, perhaps a thousand at most for a mega-site) since they need to be managed manually. The de-correlated rule sets will be somewhat larger, however even relatively small CAMs will suffice to hold the entire set.

A preferred method for cost-sensitive versions of a cryptography acceleration chip in accordance with the present invention is to implement DRAM-based classification, with a dedicated narrow DRAM port to hold classification data (i.e. a 32-bit SGRAM device). A higher performance alternative is to use external SSRAM, in which case a shared memory system can readily sustain the required low latency.

Both variants of packet classifier are described herein. The RAM-based variant, illustrated in Fig. 4 relies upon a classification entry structure in external memory. The RAM-based classifier operates via a hash-based lookup mechanism. RAM-based classification requires one table per type of match: one for IPSec quintuples, one for IPSec triples, and a small table for fragmentation lookups.

An important property of DRAM-based matching is that only exact matches are kept in the DRAM-based tables, i.e., it is not possible to directly match with wildcards and bit masks the way a CAM can. Host CPU assistance is required to dynamically map IPSec policies into exact matches. This process occurs once every time a new connection is created. The first packet from such a connection will require the creation of an exact match based on the applicable IPSec policy entry. The host CPU load created by this process is small, and can be further reduced by providing microcode assistance.

The input match fields are hashed to form a table index, which is then used to look up a Hash Map table. The output of this table contains indexes into a Classification Entry table that holds a copy of match fields plus additional match tag information.

The Hash Map and Classification Entry tables are typically stored in off-chip DRAM. Since every access to these tables involves a time-consuming DRAM fetch, a fetch algorithm which minimizes the number of rehash accesses is desirable. In most typical scenarios, a matching tag is found with just two DRAM accesses with a chip in accordance with the present invention.

To this effect, the hash table returns indexes to three entries that could match in one DRAM access. The first entry is fetched from the Classification Table; if this matches the

classification process completes. If not, the second then the third entry are fetched and tested for a match against the original match field. If both fail to match, a rehash distance from the original hash map entry is applied to generate a new hash map entry, and the process repeated a second time. If this fails too, a host CPU interrupt indicating a match failure is generated. When this occurs, the host CPU will determine if there is indeed no match for the packet, or if there is a valid match that has not yet been loaded into the classifier DRAM tables. This occurs the first time a packet from a new connection is encountered by the classification engine.

Because the hash table is split into a two-level structure, it is possible to maintain a sparse table for the top-level Hash Map entries. Doing so greatly reduces the chances of a hash collision, ensuring that in most cases the entire process will complete within two DRAM accesses.

The following code shows the Hash Map table entries as well as the Classification Entries:

```
/*
 * Security Association Table - Classification Fields
 * Used to look up an association per header.
 * This table is accessed via a hash lookup structure, SATClassHash, defined next.
 *
 * Note that a single IPSec Security Association Database entry can occupy multiple
 * SATClass entries due to wildcard and range support for various header fields.
 *
*/

typedef struct SATClass_struct {
        u32     srcAddr;                /* IP source address */
        u32     dstAddr;                /* IP destination address */
        u16     srcPort;                /* TCP source port */
        u16     dstPort;                /* TCP destination port */
        u32     spi;                    /* Security Parameter Index */
        u8      protocol;               /* Next level protocol */
        u32     tag;                    /* Match tag */
} SATClass;

/*
 * Hash table structure to look up an entry in the Security
 * Association Table Classification
 * Fields array. Each hash bucket holds up to three entries pointing
 * to sat_class values.
 * There are two hash table structures -- one for SPI-based lookup,
 * one for inner header lookup.
 *
 * Overflows are handled via software. The odds of an overflow are small -- the
 * average hash bucket occupancy is 0.5 entries per bucket,
 * and an initial overflow is handled via a variable-distance rehash.
 * Host software can set the rehash distance per hash entry to minimize
 * overflow situations. An overflow would require 3 entries in the first
 * hash bucket, followed by 3 entries in the second re-hashed
 * bucket as well. This is very unlikely in practice.
 *
 * Multiple matching SATClass entries need to searched sequentially.
*/
typedef struct SATClassHash_struct {
        /* Up to three pointers (index) of SATClass entries */
        SATClass *Index0, *Index1, *Index2;
        u32 SATPresent:10;      /* 2 low order bits are # entries (0-3) */
                        /* 8 high order bits are rehash distance */
} SATClassHash;
```

In one embodiment of the present invention, a Hash Map structure entry is 128-bits long, and a Classification Entry is 192-bits long. This relatively compact representation enables huge numbers of simultaneous security associations to be supported in high-end systems, despite the fact that DRAM-based matching requires that only exact matches be stored in memory. As an example, the DRAM usage for 256K simultaneous sessions for IPSec quintuple matches is as follows:

Classification Entry memory: 24 Bytes * 256K → 6.1 Mbytes of DRAM usage
Hash Map memory: Sparse (0.5 entries per hash bucket avg), 2 * 16 Bytes * 256K → 8M Bytes

Total DRAM usage for 256K simultaneous sessions is under 16 Mbytes; 256K sessions would be sufficient to cover a major high-tech metropolitan area, and is appropriate for super high-end concentrator systems.

Since DRAM-based classification requires one table per type of match, the total memory usage is about double the above number, with a second table holding IPSec triple matches. This brings the total up to 32Mbytes, still very low considering the high-end target concentrator system cost. A third table is needed for fragmentation lookups, but this table is of minimal size.

Another attractive solution is to use SSRAM to build a shared local memory system. Since SSRAM is well suited to the type of random accesses performed by RAM-based classification, performance remains high even if the same memory bank is used for holding both packet and classification data.

Further performance advances may be achieved using a CAM based classification engine in accordance with the present invention. The CAM based classifier is conceptually much simpler than the DRAM based version. In one embodiment, it is composed of a 104-bit match field that returns a 32-bit match tag, for a total data width of 136-bits. In contrast to DRAM-based classification, a common CAM array can readily be shared among different types of lookups. Thus a single CAM can implement all forms of lookup required by a cryptography acceleration chip in accordance with the present invention, including fragment lookups, IPSec quintuple matches, and IPSec triple matches. This is accomplished by storing along with each entry, the type of match that it corresponds to via match type field.

Because the set of IPSec rules are pre-processed via a de-correlation step and a binarization step prior to mapping to CAM entries, it is not necessary for the CAM to support any form of ordered search. Rather, it is possible to implement a fully parallel search and return any match found.

Referring to Fig. 5, the preferred implementation involves an on-chip CAM that is capable of holding 128 entries. Each entry consists of a match field of 106-bits (including a 2-bit match type code) and a match tag of 32-bits. An efficient, compact CAM implementation is desired in order to control die area. The CAM need not be fast; one match every 25 clock cycles will prove amply sufficient to meet the performance objective of one lookup every 400ns. This allows a time-iterated search of CAM memory, and allows further partitioning of CAM contents into sub-blocks that can be iteratively searched. These techniques can be used to cut the die area required for the classifier CAM memory.

CAM matching is done using a bit mask to reflect binarized range specifiers from the policy rule set. In addition, bit masks are used to choose between IPSec quintuple, triple, fragment or non-IPSec basic matches.

Should on-chip CAM capacity prove to be a limitation, an extension mechanism is provided to access a much larger off-chip CAM that supports bit masks. An example of such a device is Lara Technologies' LTI1710 8Kx136/4Kx272 ternary CAM chip.

Typical security policy rule sets range from a few entries to a hundred entries (medium corporate site) to a maximum of a thousand or so entries (giant corporate site with complex policies). These rule sets are manually managed and configured, which automatically limits their size. The built-in CAM size should be sufficient to cover typical sites with moderately complex rule sets; off-chip CAM can be added to cover mega-sites.

CAM-based classification is extremely fast, and will easily provide the required level of performance. As such, the classifier unit does not need any pipelining, and can handle multiple classification requests sequentially.

Figs. 6A and 6B provide process flow diagrams showing aspects of the inbound and outbound packet processing procedures (including lookups) associated with packet classification in accordance with one embodiment of the present invention. Fig. 6A depicts the flow in the inbound direction (600). When an inbound packet is received by the packet classifier on a cryptography acceleration chip in accordance with the present invention, its header is parsed (602) and a SAD lookup is performed (604). Depending on the result of the SAD lookup and as specified by the resulting policy, the packet may be dropped (606), passed-through (608), or directed into the cryptography processing system. Once in the system, the packet is decrypted and authenticated (610), and decapsulated (612). Then, a SPD lookup is performed (614). If the result of the lookup is a policy that does not match that specified by the SAD lookup, the packet is dropped (616). Otherwise, a clear text packet is sent out of the cryptography system (618) and into the local system/network.

Fig. 6B depicts the flow in the outbound direction (650). When an outbound packet is received by the packet classifier on a cryptography acceleration chip in accordance with the present invention, its header is parsed (652) and a SPD lookup is performed (654). Depending on the result of the SPD lookup and as specified by the resulting policy, the packet may be dropped (656), passed-through (658), or directed into the cryptography processing system. Once in the system, a SAD lookup is conducted (660). If no matching SAD entry is found (662) one is created (664) in the IPSec Security Association Database. The packet is encapsulated (666), encrypted and authenticated (668). The encrypted packet is then sent out of the system (670) to the external network (WAN).

Examples

The following examples describe and illustrate aspects and features of specific implementations in accordance with the present invention. It should be understood the following is representative only, and that the invention is not limited by the detail set forth in these examples.

*Example 1: Security Association Prefetch Buffer*

The purpose of the SA buffer prefetch unit is to hold up to eight Security Association Auxiliary structures, two per active processing engine. This corresponds to up to two packet processing requests per engine, required to support the double-buffered nature of each engine. The double buffered engine design enables header prefetch, thus hiding DRAM latency from the processing units. The structures are accessed by SA index, as generated by the packet classifier.

Partial contents for the SA Auxiliary structure are as shown in the following C code fragment:

```
typedef struct SATAux_struct {
        u32 byteCount;          /* Total payload bytes processed via */
                                /* this entry (larger of crypto or auth bytes) */
        u64 expiry;             /* Expiry time or #bytes for this */
                                /* entry (checked per use) */
        u32 packetCount;        /* Stats - # packets processed via this entry */
        struct SATAux_struct *next;   /* Next IPSec Security Association for SA */
                                      /* bundles */
        u32 seqNoHi;            /* Anti replay sequence number - "right" edge of window */
                                /* for outgoing packets, used for next sequence number */
        u64 seqWin;             /* Anti-replay sequence window (bit mask) */
        u32 peerAddr;           /* IPSec peer security gateway address */
        u32 spi;                /* IPSec security parameter index */
        u8 originalProtocol;/* pre-IPSec Protocol to which this SA applies */

        cryptoState algoCrypto; /* Keys and other parameters for crypto */
        authState algoAuth;       /* Keys, state and other HMAC parameters */

        u8 enableSeq:1;         /* 1 to enable anti-replay sequence check */
        u8 crypto:2;            /* DES, 3DES, RC4, NONE */
        u8 auth:2;              /* MD5, SHA1, NONE */
        u8 format:2;            /* FORMAT_ESP, FORMAT_AH, FORMAT_AH_ESP */
        u8 tunnel:1;            /* 1 to enable tunneling, 0 to use transport adjacency */
        u8 discard:1;           /* Drop packet */
        u8 pass:1;              /* Pass packet through */
        u8 intr:1;              /* Interrupt upon match to this entry */
                                /* (useful for drop/pass) */
        u8 explicitiv:1;        /* Use implicit IV from SAdB as opposed to explicit */
                                /* IV from packet */
        u8 padnull:1;           /* Apply pad to 64-byte boundary for ESP */
                                /* null crypto upon IPSec output */
        u8 oldpad:1;            /* Old style random padding per RFC1829 */
} SATAux;
```

The SA Buffer unit prefetches the security auxiliary entry corresponding to a given SA index. Given an SA index, the SA buffer checks to see if the SA Aux entry is already present; if so, an immediate SA Hit indication is returned to the distributor micro-engine. If not, the entry is pre-fetched, and a hit status is then returned. If all SA entries are dirty (i.e. have been previously written but not yet flushed back to external memory) and none of the entries is marked as retired, the SA Buffer unit stalls. This condition corresponds to all processing engines being busy anyway, such that the distributor is not the bottleneck in this case.

*Example 2: Distributor Microcode Overview*

In one implementation of the present invention, the distributor unit has a micro-engine large register file (128 entries by 32-bits), good microcode RAM size (128 entries by 96-bits), and a simple three stage pipeline design that is visible to the instruction set via register read delay slots and conditional branch delay slots. Microcode RAM is downloaded from the system port at power-up time, and is authenticated in order to achieve FIPS140-1 compliance. In order to ensure immediate micro-code response to hardware events, the micro-engine is started by an event-driven mechanism. A hardware prioritization unit automatically vectors the micro-engine to the service routing for the next top-priority outstanding event; packet retiring has priority over issue.

```
Packet Issue Microcode:

    //
    // SA Buffer entry has been pre-fetched and is on-chip
    // Packet length is available on-chip
    //
    test drop/pass flags; if set special case processing;
    test lifetime; break if expired;     // reset if auth fails later
    test byte count; break if expired;   // reset if auth fails later
    assert stats update command;         // update outgoing sequence number
    assert locate next engine command; if none, stall;
    assert issue new packet command with descriptor ID, tag, length;
```

Since the distributor unit is fully pipelined, the key challenge is to ensure that any given stage keeps up with the overall throughput goal of one packet every 50 clock cycles. This challenge is especially important to the micro-engine, and limits the number of micro-instructions that can be expended to process a given packet. The following pseudo-code provides an overview of micro-code functionality both for packet issue and for packet retiring, and estimate the number of clock cycles spent in distributor micro-code.

```
Packet Retiring Microcode:

    //
    // SA Buffer entry has been pre-fetched and is on-chip
    // Packet length is available on-chip. Packet has been authenticated
    // by now if authentication is enabled for this flow.
    //
    if sequence check enabled for inbound, check & update sequence mask;
    update Engine scheduling status;
    mark packet descriptor as free; add back to free pool; // Schedule write
```

Since most distributor functions are directly handled via HW assist mechanisms, the distributor microcode is bounded and can complete quickly. It is estimated that packet issue will require about 25 clocks, while packet retiring will require about 15 clocks, which fits within the overall budget of 50 clocks.

*Example 3: Advanced Classification Engine (ACE)*

In one specific implementation of the present invention, a classification engine (referred to as the Advanced Classification Engine (ACE)) provides an innovative solution to the difficult problem of implementing the entire set of complex IPSec specified Security Association Database and Security Policy Database rules in hardware. The IETF IPSec protocol provides packet classification via wildcard rules, overlapping rules and conflict resolution via total rule ordering. The challenge solved by ACE is to implement this functionality in wirespeed hardware.

The Advanced Classification Engine of a chip in accordance with the present invention handles per-packet lookup based on header contents. This information then determines the type of IPSec processing that will be implemented for each packet. In effect, ACE functions as a complete hardware IPSec Security Association Database lookup engine. ACE supports full IPSec Security Association lookup flexibility, including overlapping rules, wildcards and complete ordering. Simultaneously, ACE provides extremely high hardware throughput. In addition, ACE provides value-added functions in the areas of statistics gathering and maintenance on a flexible per link or per Security Association basis, and SA lifetime monitoring. A separate unit within ACE, the Automatic Header Generator, deals with wirespeed creation of IPSec compliant headers.

ACE derives its extremely high end to end performance (5 Mpkt/s at 125MHz) from its streamlined, multi-level optimized design. The most performance critical operations are handled via on-chip hardware and embedded SRAM memory. The next level is handled in hardware, but uses off-chip DRAM memory. The slowest, very infrequent frequent level of operations is left to host processor software. Key features of ACE include:

- Full support for IPSec Security Association Database lookup, including wildcard rules, overlapping rules, and complete ordering of database entries.

- Extremely high hardware throughput: Fully pipelined non-blocking out-of-order design. Four datagrams can be processed simultaneously and out of order to keep throughput at full rated wirespeed.

- Flexible connection lookup based on src/dst address, src/dst ports, and protocol. Any number of simultaneously active packet classification values can be supported.

- Hardware support for header generation for IPSec Encapsulating Security Protocol (ESP) and for IPSec Authentication Header (AH).

- Full hardware header generation support for Security Association bundling – transport adjacency, and iterated tunneling.

- Sequence number generation and checking on-chip.

- Classification engine and statistics mechanisms available to non-IPSec traffic as well as to IPSec traffic.

- Security Association lifetime checking based on byte count and elapsed wall clock time.

- High quality random number generator for input to cryptography and authentication engines.

The input to ACE consists of *packet classification fields*: src/dst address, src/dst ports, and protocol. The output of ACE is an IPSec Security Association matching entry, if one exists, for this classification information within the IPSec Security Association Database. The matching entry then provides statistics data and control information used by automatic IPSec header generation.

A global state flag controls the processing of packets for which no matching entry exists – silent discard, interrupt and queue up packet for software processing, or pass through.

The matching table (SAT, Security Association Table) holds up to 16K entries in DRAM memory. These entries are set up via control software to reflect IPSec Security Association Database (SAdB) and Security Policy Database (SPdB) rules. The wildcard and overlapping but fully ordered entries of the SAdB and SPdB are used by control software to generate one non-overlapping match table entry for every combination that is active. This scheme requires software intervention only once per new match entry.

Fig. 7 shows a block diagram of the ACE illustrating its structure and key elements. Major components of ACE are as follows:

- Security Association Table Cache – Classification Field (SATC-CL): Used to look up a packet's classification fields on-chip. Each entry has the following fields:

| *SATC-CL SATC Classification Field Cache* | | | |
|---|---|---|---|
| **Field name** | **Description** | **IPv6 size (bits)** | **IPv4 size (bits)** |
| src@ | IP source address | 128 bits | 32 bits |
| dst@ | IP destination address | 128 bits | 32 bits |
| protocol | High level protocol field | 8 bits | |
| src port | High level protocol source port | 16 bits | 16 bits |
| dst port | High level protocol destination port | 16 bits | 16 bits |
| Aux field ptr | Pointer to auxiliary data (stats, lifetime) | 16 bits | |
| peer@ | IP address of IPSec peer gateway | 128 bits | 32 bits |
| spi | IPSec Security Parameter Index | 32 bits | |

| | | |
|---|---|---|
| ipsec format | ESP, AH or none; Tunnel or Adj | 3 bits |

- Security Association Auxiliary Data table Cache (SATC-AUX): Serves to hold statistics, etc. information on-chip in flexible fashion. An entry within SATC-AUX can serve multiple classification fields, allowing multiple combinations to be implemented for stats gathering. Each entry has the following fields:

| *SATC-AUX SATC Auxiliary Field Cache* | | | |
|---|---|---|---|
| **Field name** | **Description** | **IPv6 size (bits)** | **IPv4 size (bits)** |
| | | | |
| Byte count | Total byte count for this entry | 32 bits | |
| Expiry time | Time entry expires | 32 bits | |
| #misses | SATC-CL misses for this entry | 32 bits | |
| #pkt | Total packet count for this entry | 32 bits | |
| next_spi | Next SPI for Iterated tunneling or Transport adjacency | 32 bits | |
| seqchk | Enable anti-replay sequence check | 1 bit | |
| seqno | Sequence number (output) or highest received seq number (input) | 32 bits | |
| seqmask | Anti-Replay window | 64 bits | |
| algo_info | Algorithm specific data (keys, pad lengths, Initial Vectors, etc) | 296 bits | |

- Quad Refill Engine: handles the servicing of SATC-CL misses. When ever a miss occurs, the corresponding entry in the SATC-AUX is simultaneously fetched in order to maintain cache inclusion of all SATC-AUX entries within SATC-CL entries. This design simplifes and speeds up the cache hit logic considerably. The refill engine accepts and processes up to 4 outstanding miss requests simultaneously.

- Quad Header Buffers: Holds up to 4 complete IPv4 headers, and up to 256 bytes each of 4 IPv6 headers. Used to queue up headers that result in SATC-CL misses. Headers that result in a cache hit are immediately forwarded for IPSec header generation.

- Header streaming buffer: Handles overflows from the header buffer by streaming header bytes directly from DRAM memory; it is expected that such overflows will be exceedingly rare. This buffer holds 256 bytes.

- Header/Trailer processing and buffer: For input datagrams, interprets and strips IPSec ESP or AH header. For output datagrams, adjusts and calculates header and trailer fields. Holds a complete IPv4 fragment header, and up to 256 bytes of an IPv6 header. Requires

input from the cryptography modules for certain fields (authentication codes, for instance).

In addition to the above components, two data structures in DRAM memory are used by ACE for efficient operation. These are:

- Complete Security Association Table– Classification Field (SAT-CL): holds classification data. This table backs up the on-chip SAT-CL Cache. Each entry is 475 bits aligned up to 60 bytes.

- Complete Security Association Auxiliary Data table (SAT-AUX): holds auxiliary data. This table backs up the on-chip SAT-AUX Cache. Each entry is 617 bits, plus up to 223 bits of algorithm specific state (such as HMAC intermediate state), for a total of 105 bytes.

The following pseudo-code module describes major ACE input processing (received datagrams) operation:

```
Input Processing () { /* Received datagram */
     Calculate hash value based upon
          (dst@,spi,protocol);
     /* Re-hash via predetermined sequence if collision occurs */
     Lookup field in Security Association Classification Cache;
     if (no match found) {
          /* Refill cache from DRAM memory */
          Calculate new hash for DRAM entry;
          /* Re-hash in case of collision */
          /*
           * Out-of-order non-blocking execution
          */
          Schedule DRAM access (up to 4 outstanding fill req's);
          Move on to Input Processing () of next datagram;
          /* When DRAM refill has completed */
          Lookup field in DRAM Security Association table;
          Pre-fetch DRAM Auxiliary table entry;
     }
     if (no match found) {
          /*
           * Datagram does not have a SAdB entry;
           * Process based on global flags.
          */
          if (nomatch_discard) silently drop packet;
          else if (nomatch_pass) send insecure packet out;
          else queue up packet and raise interrupt;
     }
     /* Datagram has a matching SAdB entry */
     Sanity check packet header fields, including protocol;
     Verify packet data against SAdB entry;
     if (seqchk) Perform anti-replay check;
     Perform lifetime check;
     Update statistics information;
     /* Aggressive writeback to minimize future miss latency */
     Schedule SATC-AUX entry for writeback to DRAM;
     Extract SAdB processing control & crypto parameters;
     Implement SAdB-specified processing on datagram;

     /* Double check packet SAdB match as soon as possible */
     Perform SAdB lookup procedure on
          (src@,dst@,srcport,dstport,protocol);
     Verify that original SPI is returned;
}
```

The following pseudo-code module describes major ACE output processing

```
Output Processing () { /* Received datagram */
     Calculate hash value based upon
           (src@,dst@,srcport,dstport,protocol);
     /* Re-hash via predetermined sequence if collision occurs */
     Lookup field in Security Association Classification Cache;
     if (no match found) {
           /* Refill cache from DRAM memory */
           Calculate new hash for DRAM entry;
           /* Re-hash in case of collision */
           /*
            * Out-of-order non-blocking execution
           */
           Schedule DRAM access (up to 4 outstanding fill req's);
           Move on to Input Processing () of next datagram;
           /* When DRAM refill has completed */
           Lookup field in DRAM Security Association table;
           Pre-fetch DRAM Auxiliary table entry;
     }
     if (no match found) {
           /*
            * Datagram does not have a SAdB entry; process based
            * Process based on global flags.
           */
           if (nomatch_discard) silently drop packet;
           else if (nomatch_pass) send insecure packet out;
           else queue up packet and raise interrupt;
     }
     /* Datagram has a matching SAdB entry */
     Sanity check packet header fields, including protocol;
     Generate sequence number;
     Perform lifetime check;
     Update statistics information;
     /* Aggressive writeback to minimize future miss latency */
     Schedule SATC-AUX entry for writeback to DRAM;
     Extract SAdB processing control & crypto parameters;
     Implement SAdB-specified processing on datagram;
}
```

(transmitted datagrams) operation:

ACE implements multiple techniques to accelerate processing. The design is fully pipelined, such that multiple headers are in different stages of ACE processing at any given time. In addition, ACE implements non-blocking out-of-order processing of up to four packets.

Out of order non-blocking header processing offers several efficiency and performance enhancing advantages. Performance-enhancing DRAM access techniques such as read combining and page hit combining are used to full benefit by issuing multiple requests at once to refill SATC-CL and SATC-AUX caches. Furthermore, this scheme avoids a problem similar to Head Of Line Blocking in older routers, and minimizes overall packet latency.

Because of the pipelined design, throughput is gated by the slowest set of stages.

| | |
|---|---|
| Header parsing | 2 clocks |
| Hash & SA Cache lookup | 2 clocks |
| Hash & SA Auxiliary lookup | 2 clocks |

| | |
|---|---|
| Initial header processing, anti-replay | 4 clocks |
| Statistics update | 3 clocks |
| Final header update | 6 clocks |

This works out to 19 clocks per datagram total with zero pipelining, within a design goal of 25 clocks per packet (corresponding to a sustained throughput of 5Mpkt/s at 125MHz). A simple dual-stage pipeline structure is sufficient, and will provide margin (average throughput of 10 clocks per header). The chip implements this level of pipelining.

ACE die area is estimated as follows based on major components and a rough allocation for control logic and additional data buffering:

| | |
|---|---|
| Control logic overhead | 50Kg |
| Quad header buffer | 20Kg |
| Quad refill controller with tag match | 50Kg |
| SATC-CL cache | 130Kg (single port) |
| SATC-AUX cache | 170Kg (single port) |
| Stats engine | 10Kg |
| Header/Trailer processor | 20Kg |
| Prefetch buffering | 50Kg |

Total estimated gate count is 500Kg.

References

The following references, which provide background and contextual information relating to the present invention, are incorporated by reference herein in their entirety and for all purposes:

"Efficient Fair Queuing using Deficit Round Robin", M. Shreedhar, G. Varghese, October 1996.

draft-ietf-pppext-mppe-03.txt Microsoft Point-To-Point Encryption (MPPE) Protocol, G. S. Pall, G. Zorn, May 1999

draft-ietf-nat-app-guide-02.txt "NAT Friendly Application Design Guidelines", D. Senie, September 1999.

draft-ietf-nat-rsip-ipsec-00.tx] "RSIP Support for End-to-end IPSEC", G. Montenegro, M. Borella, May 19 1999.

draft-ietf-ipsec-spsl-01.txt, "Security Policy Specification Language", M. Condell, C. Lynn, J. Zao, July 1, 1999

"Random Early Detection Gateways for Congestion Avoidance", S. Floyd, V. Jacobson, August 1993 ACM Transactions on Networking

"The IP Network Address Translator (NAT)", K. Egevang, P. Francis, May 1994.

"DEFLATE Compressed Data Format Specification version 1.3", P. Deutsch, May 1996.

"Specification of Guaranteed Quality of Service", S. Shenker, C. Partridge, R. Guerin, September 1997.

"IP Network Address Translator (NAT) Terminology and Considerations", P. Srisuresh, M. Holdrege, August 1999.

"IP Payload Compression using DEFLATE", R. Pereira, December 1998.

S. Kent, R. Atkinson, "Security Architecture for the Internet Protocol," RFC 2401, November 1998 (obsoletes RFC 1827, August 1995).

S. Kent, R. Atkinson, "IP Authentication Header," RFC 2402, November 1998 (obsoletes RFC 1826, August 1995).

S. Kent, R. Atkinson, "IP Encapsulating Payload," RFC 2406, November 1998 (obsoletes RFC 1827, August 1995).

Maughhan, D., Schertler, M., Schneider, M., and Turner, J., "Internet Security Association and Key Management Protocol (ISAKMP)," RFC 2408, November 1998.

Harkins, D., Carrel, D., "The Internet Key Exchange (IKE)," RFC 2409, November 1998.

"Security Model with Tunnel-mode IPsec for NAT Domains", P. Srisuresh, October 1999.

"On the Deterministic Enforcement of Un-Ordered Security Policies", L. Sanchez, M. Condell, February 14th 1999.

## CONCLUSION

Although the foregoing invention has been described in some detail for purposes of clarity of understanding, those skilled in the art will appreciate that various adaptations and

modifications of the just-described preferred embodiments can be configured without departing from the scope and spirit of the invention. For example, other cryptography engines may be used, different system interface configurations may be used, or modifications may be made to the packet processing procedure. Moreover, the described processing distribution and classification engine features of the present invention may be implemented together or independently. Therefore, the described embodiments should be taken as illustrative and not restrictive, and the invention should not be limited to the details given herein but should be defined by the following claims and their full scope of equivalents.

*What is claimed is:*